Lecture 8: Search 7

Victor R. Lesser

CMPSCI 683
Fall 2010

This Lecture

- Continuation of Local Search
  - Hill-Climbing/Iterative Improvement
    - Simulated Annealing (Stochastic Hill Climbing)
  - Beam Search
    - Genetic Algorithm

Iterative Improvement Algorithms

- What is the search space
  - Search space of complete solutions vs. partial solutions
- When useful:
  - “Reasonable” complete solution can be generated
    - Importance of good starting point
  - Operator to modify complete solution
  - Some notion of progress
    - Measure of complete solution in terms of constraint violations or an evaluate function

Example: 4 queens

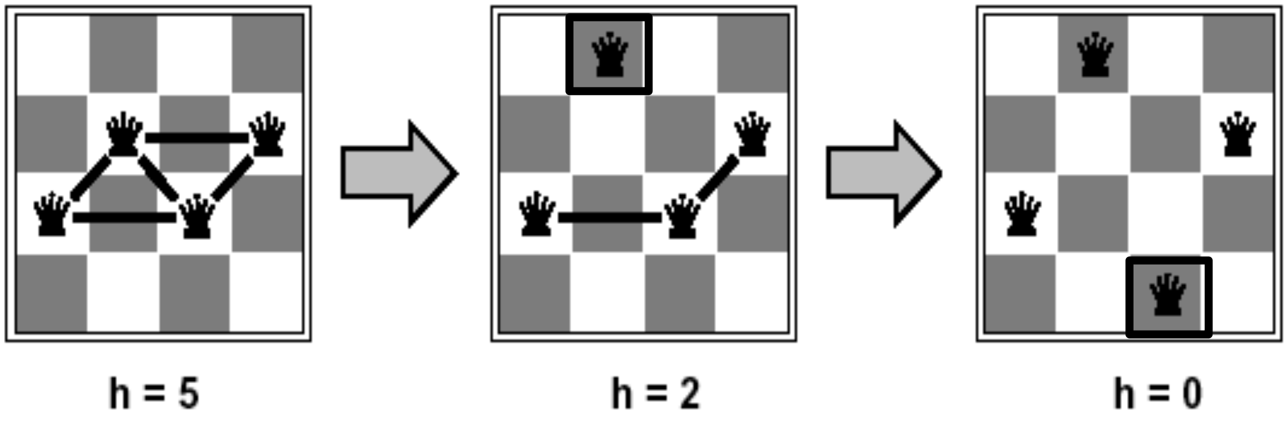
States: 4 queens in 4 columns ($4^4 = 256$ states)

Operators: move queen in column

Goal test: no attacks

Evaluation: $h(n)$ = number of attacks

- Almost always solves very large problems almost instantly (e.g., n = 1 million)



Hill-Climbing(HC) Search

- The main iterative improvement algorithm is hill-climbing:

Continually move in the direction of increasing value of all successor states until a maximum is reached.

- This is sometimes called **steepest-ascent** HC, and is called **gradient descent** search if the evaluation function is based on minimizing cost rather than maximizing quality.

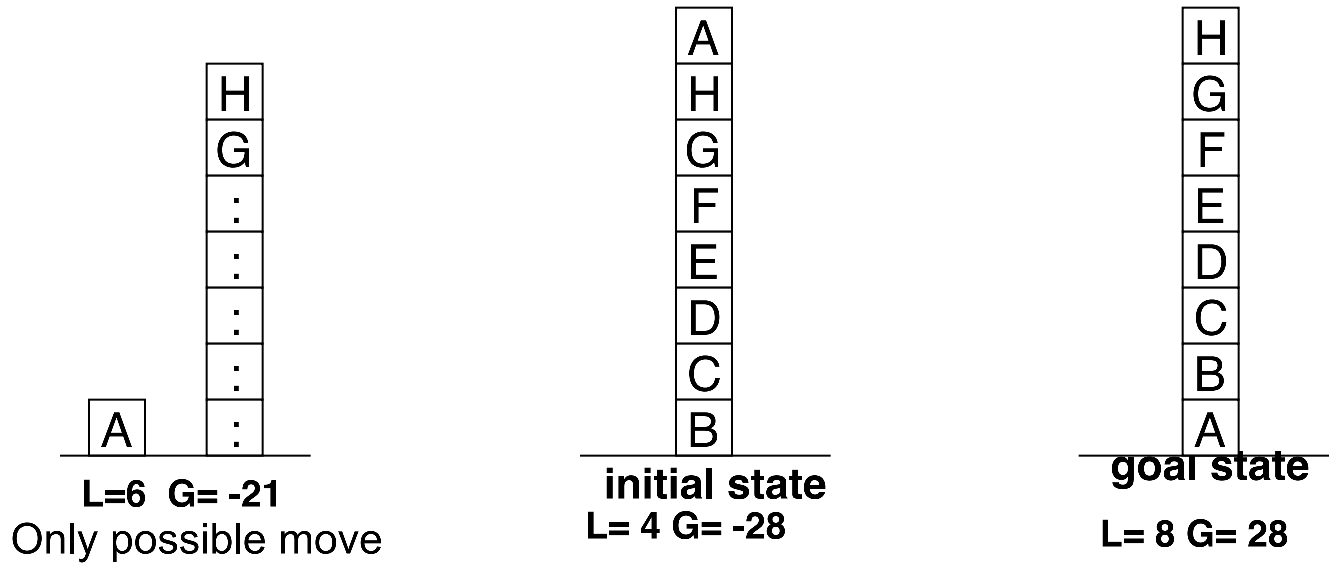
Steepest Ascent Hill-Climbing

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] < VALUE[current] then return STATE[current]
        current ← neighbor
    end
```

Looks at all successors

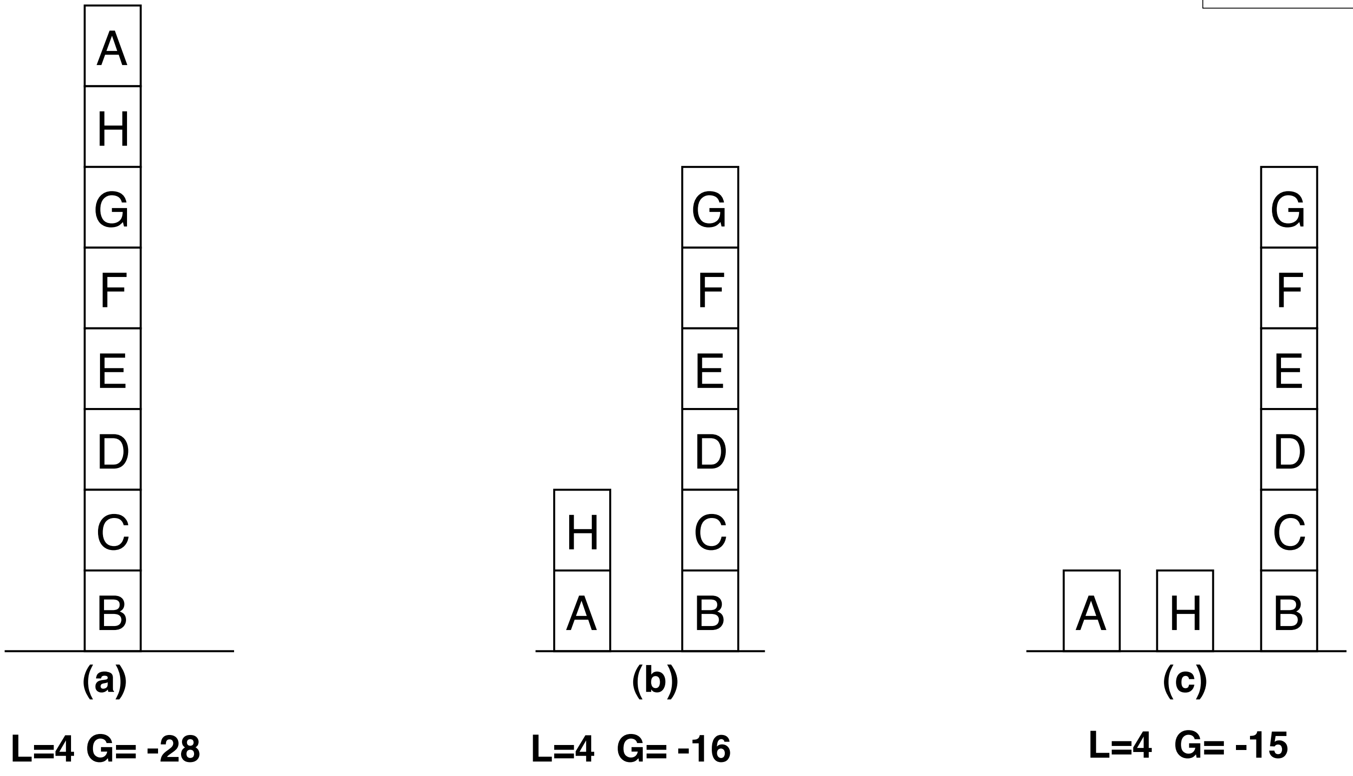
An Example of Hill-Climbing Problems



L(local) evaluation function: Add 1 point for every block that is resting on the thing it is supposed to be resting on. Subtract 1 point for every block that is sitting on the wrong thing.

Gl(gobal) evaluation function : For each block that has the correct support structure (i.e., the complete structure underneath it is exactly as it should be), add 1 point for every block in the support structure. For each block that has an incorrect support structure, subtract one point for every block in the existing support structure.

An Example of Hill-Climbing Problems (cont'd) Next Possible Moves

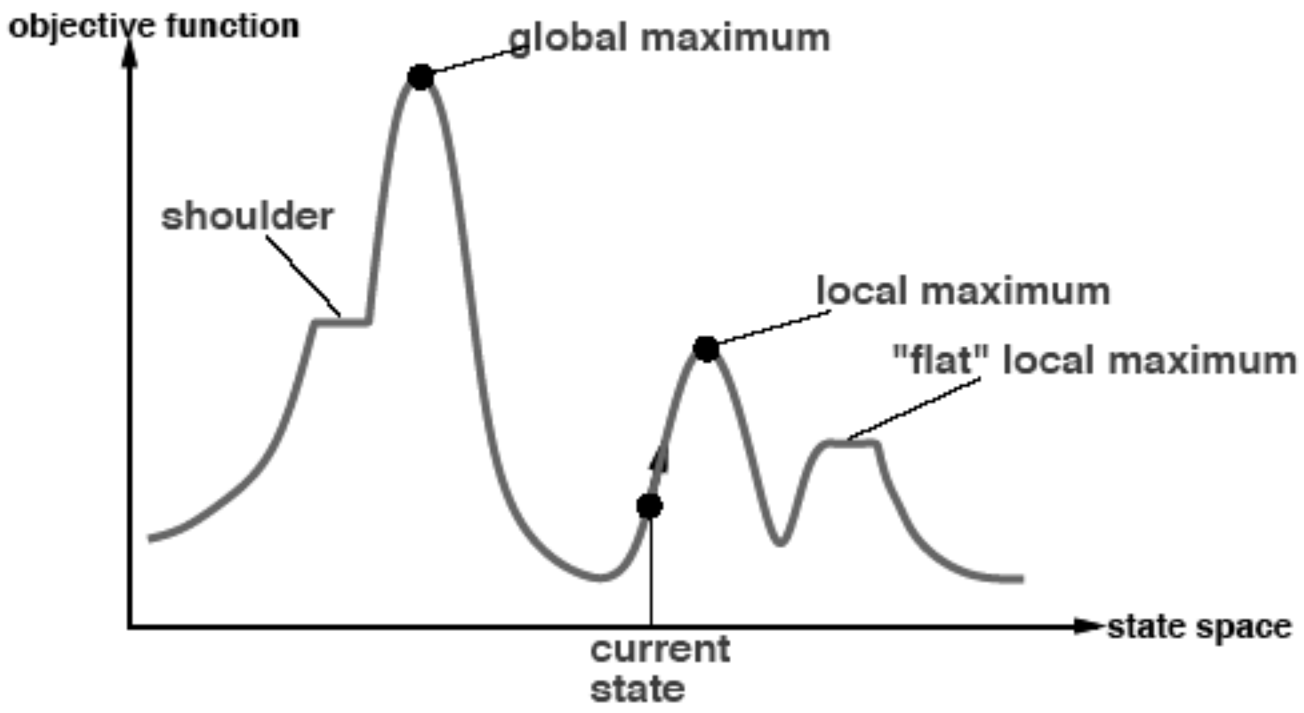


Local criterion results in no available moves that increase evaluation

Hill-climbing search

A simple form of local search: Continually move in the direction of increasing value.

- Greedy Local search; grabs good neighbor without thinking ahead

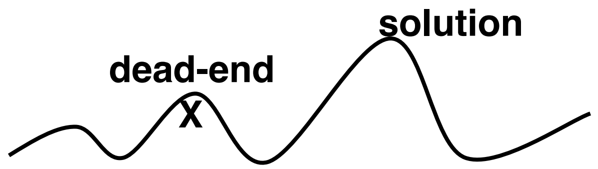


Useful to consider what happened at different points in the state space landscape

Problems with Hill Climbing

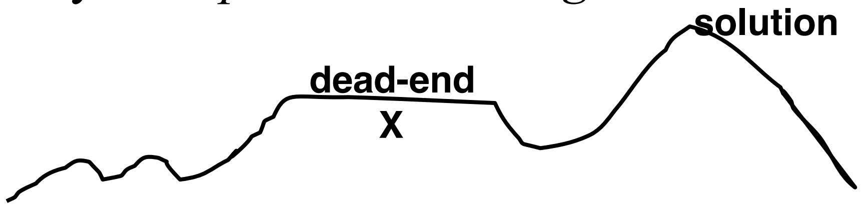
- Can get stuck at a *local maximum*.

Local maximum



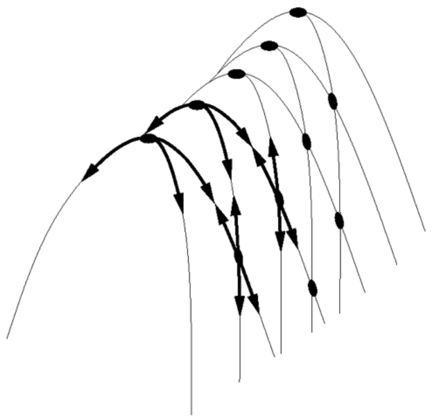
- Unable to find its way off a *plateau with single moves*.

Plateau



- Cannot climb along a narrow *ridge* when almost all steps go down (continuous space).

Ridge/Knife edges



Especially serious when you can not evaluate all potential moves

Variants of hill-climbing

Ways to overcome the weaknesses:

- **Stochastic hill-climbing (Simulated Annealing)**: choose at random an uphill move among the successors
  - *Sometimes take downhill steps*
  - *You may have to get worse to get better!!*
- **First-choice hill climbing**: generate successors randomly until finding an uphill move
- **Random-restart hill climbing**: restart search from randomly generated initial states if not making progress

Simulated Annealing

"Simulated annealing is a variation of hill climbing in which, at the beginning of the process, some downhill moves may be made. The idea is to do enough exploration of the whole space early on so that the final solution is relatively insensitive to the starting state. This should lower the chances of getting caught at a local maximum, a plateau, or a ridge."

Simulated annealing search

- Generate successors randomly
- Allow "bad" moves with some probability $e^{\Delta E/T}$
  - Proportional to the value (or "energy") difference ΔE
  - Modulated by a "temperature" parameter T
- Gradually decrease the frequency of such moves and their size by reducing the "temperature"
  - more time goes, less willing to explore non-optimal path
- Originated in modeling physical processes
- Optimal when T is decreased "slowly"

Simulated Annealing Algorithm

1. Evaluate the initial state. If it is also a goal state, then return it and quit. Otherwise, continue with the initial state as the current state.

2. Initialize *BEST-SO-FAR* as the current state.

3. Initialize *T* according to the annealing schedule

Simulated Annealing Algorithm (cont'd)

4. Loop until a solution is found or until there are no new operators left to be applied in the current state.

   a) Select an operator (randomly) that has not yet been applied to the current state and apply it.

   b) Evaluate the new state. Compute
   ΔE = (value of current) - (value of new state)
   - **If the new state is a goal state, return it and quit.**
   - **If it is not a goal state but is better than the current state, then make it the current state.**
     - **set *BEST-SO-FAR* to this new state if better than current *BEST-SO-FAR* .**
   - **If it is not better than the current state, then make it the current state with probability *p´*= f(**ΔE,T)**.**

   c) Revise *T* according to the annealing schedule

5. Return *BEST-SO-FAR* as the answer. ; anytime property

Simulated Annealing Algorithm

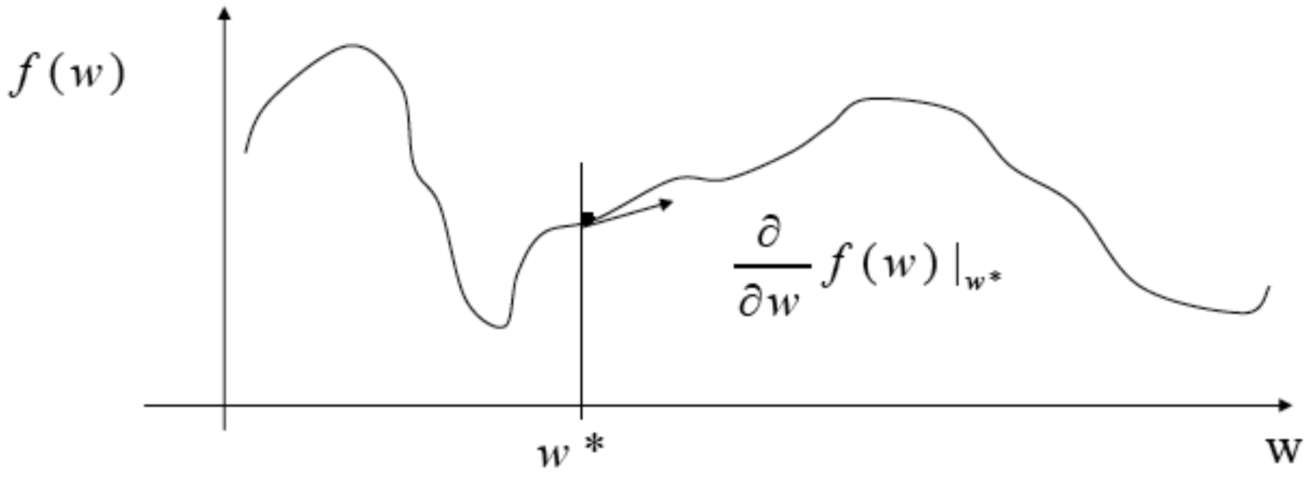
Idea: escape local maxima by allowing some "bad" moves
but gradually decrease their size and frequency

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    $T$, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for $t \leftarrow$ 1 to $\infty$ do
        $T \leftarrow$ schedule[$t$]
        if $T = 0$ then return current
        next ← a randomly selected successor of current
        $\Delta E \leftarrow$ VALUE[next] – VALUE[current]
        if $\Delta E > 0$ then current ← next
        else current ← next only with probability $e^{\Delta E/T}$
```

Gradient ascent methods

- **Gradient ascent:** the same as hill-climbing, but in the continuous parametric space **w**



- Change the parameter value of w according to the gradient

$$w \leftarrow w^* + \alpha \frac{\partial}{\partial w} f(w)|_{w^*}$$

Beam Search vs Random-Restart Hill Climbing

- Keep track of K states rather than just one
  - Modified breadth-first, contour created dynamically
- Start with K randomly generated states
- Stop if any goal state
- Multiple Successors generated for each of the k states
- Choose top K successor states in next cycle
- Contrast with Random Restart
  - **Positive** -- Sharing information across different searches by choosing top K successor states
  - **Negative** - May eliminate diversity coming from random starting points

Evolutionary Computation

- Beam Search patterned after biological evolution
  - Learning as Search
- Metaphor of Natural Selection
  - Offspring are similar to their parents
  - The more fit are more likely to have children
  - Occasionally random mutations occur

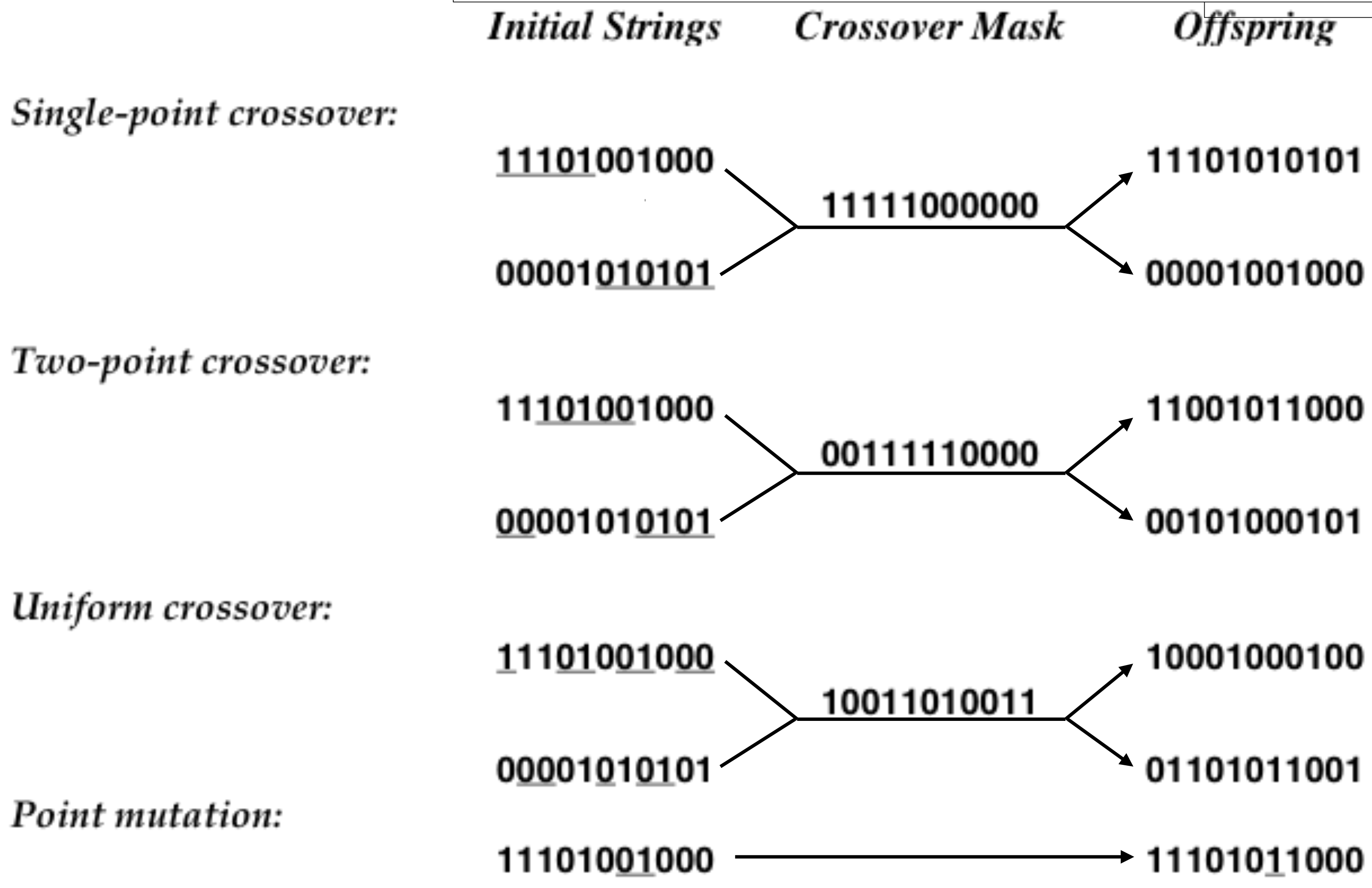
Genetic (Search) Algorithms

- Localized Beam Search
  - Specialized approach for generating successors and for selecting next states
  - **An individual solution is represented by a sequence of “genes**”.
  - The selection strategy is randomized with probability of selection proportional to “fitness”.
  - Individuals selected for reproduction are randomly paired, certain genes are crossed-over, and some are mutated.

Basic Operation of Genetic Search

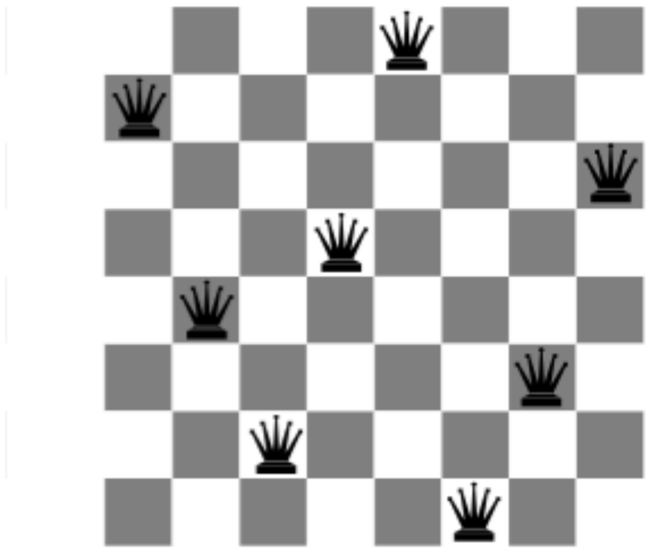
- Selection
  - More fit members are likely to be in next generation
- Mutation
  - Random altering of characteristics
- Crossover
  - Combine two members of population
    - Cross-over is probably the key idea
    - ***Exploit relative independence of certain subproblem solutions imbedded in different members***

Operators for Genetic Algorithms



Key Questions for Genetic Search

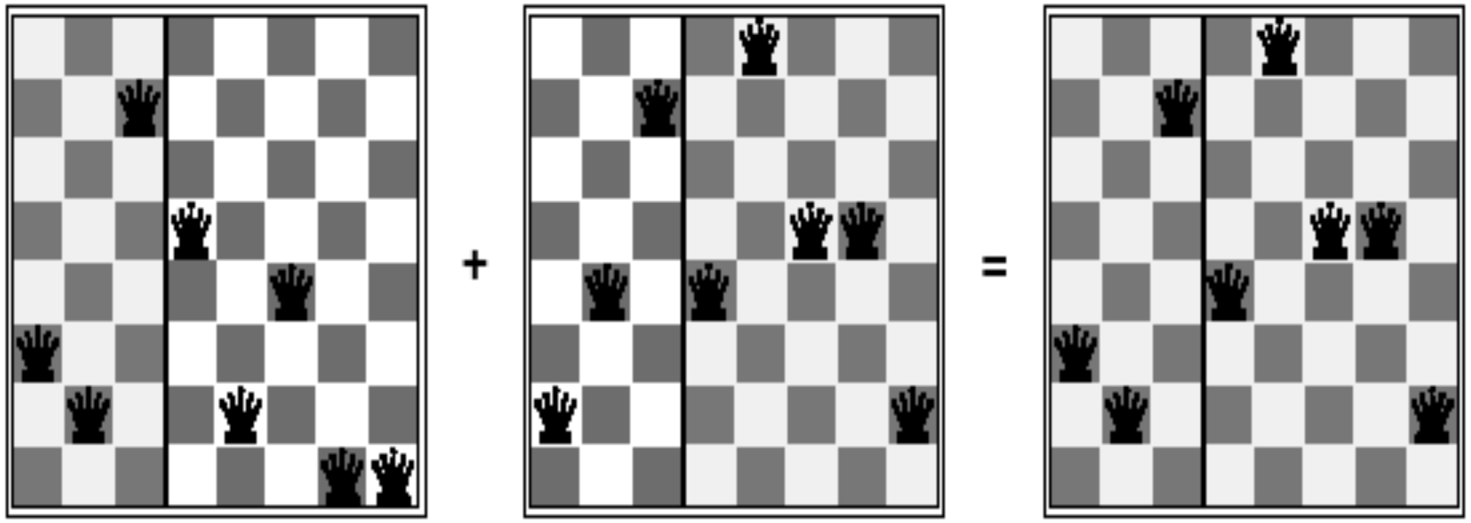
- What is the fitness function?
- How is an individual represented?
- How are individuals selected?
- How do individuals reproduce?



Genetic Algorithms Example

GAs require states encoded as strings (GPs use programs)

Crossover helps iff substrings are meaningful components



GA (*Fitness, Fitness-threshold, p, r, m*)

- GA (*Fitness, Fitness-threshold, p, r, m*)
  - Initialize: P ← p random hypotheses
  - Evaluate for each h in P, *compute-fitness (h)*
  - While [$\max_h$ *fitness (h)*] < *Fitness-threshold*
    1. *Select probabilistically select (1-r)· p members to add to P_s*

       Pr (h_i)= fitness (h_i)/ Sum of fitness of all members of P

    2. *Crossover: probabilistically select r·p/2 pairs of hypotheses from P. For each pair (h_j,h_k), produce two offspring by applying the Crossover operator. Add all offspring to P_s*

...continued from previous slide

3. ***Mutate:*** **Invert a randomly selected bit in $m \cdot p$ random members of P_s**

4. ***Update:*** **$P \leftarrow P_s$**

5. ***Evaluate:*** **for each *h* in *P*, compute *Fitness(h)***

- **Return the hypothesis from *P* that has the highest fitness.**

Selecting Most Fit Hypotheses

***Fitness proportionate selection*:**

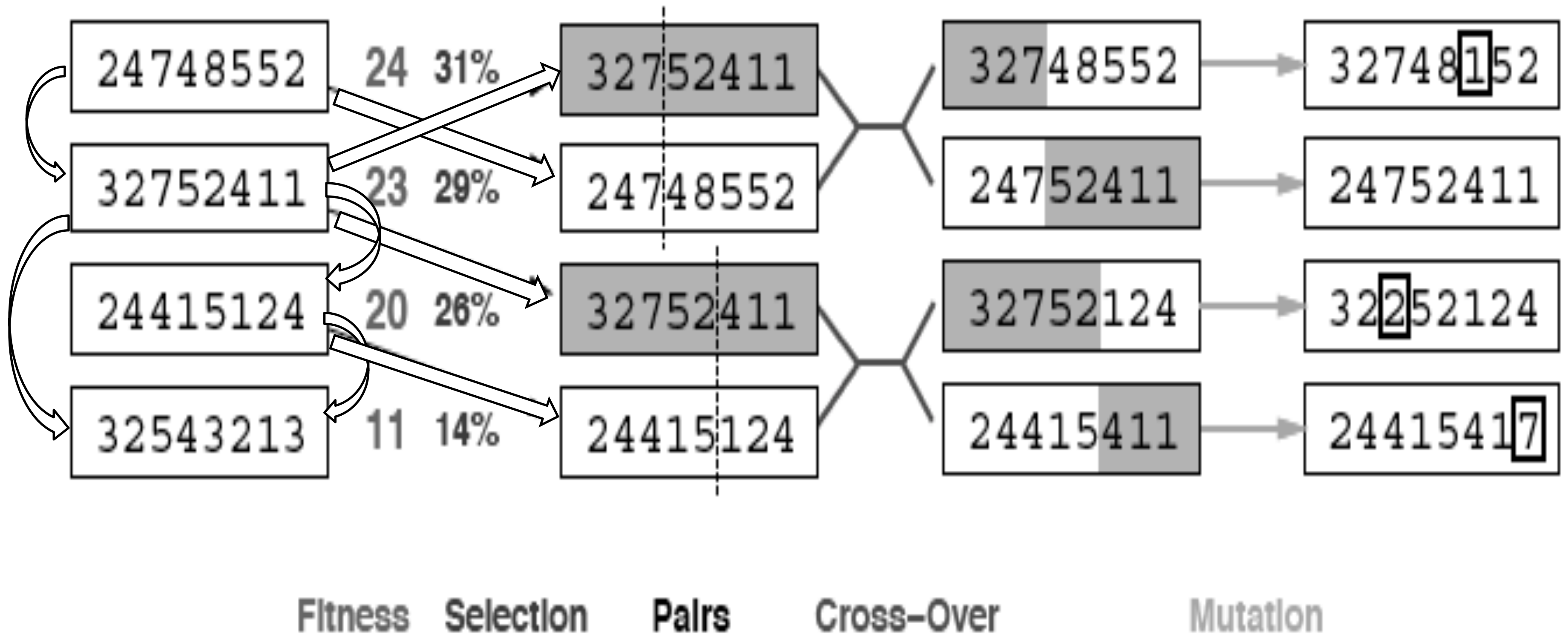
Pr (h_i)= *fitness* (h_i)/ *Sum of fitness of all members of P*

…can lead to *crowding (lack of diversity)*

- *Tournament Selection*:
  - Pick h_1, h_2 at random with uniform probability.
  - With probability p, select the more fit
- *Rank Selection*:
  - Sort all hypotheses by fitness
  - Probability of selection is proportional to rank

Genetic Algorithms Example

= stochastic local beam search + generate successors from pairs of states

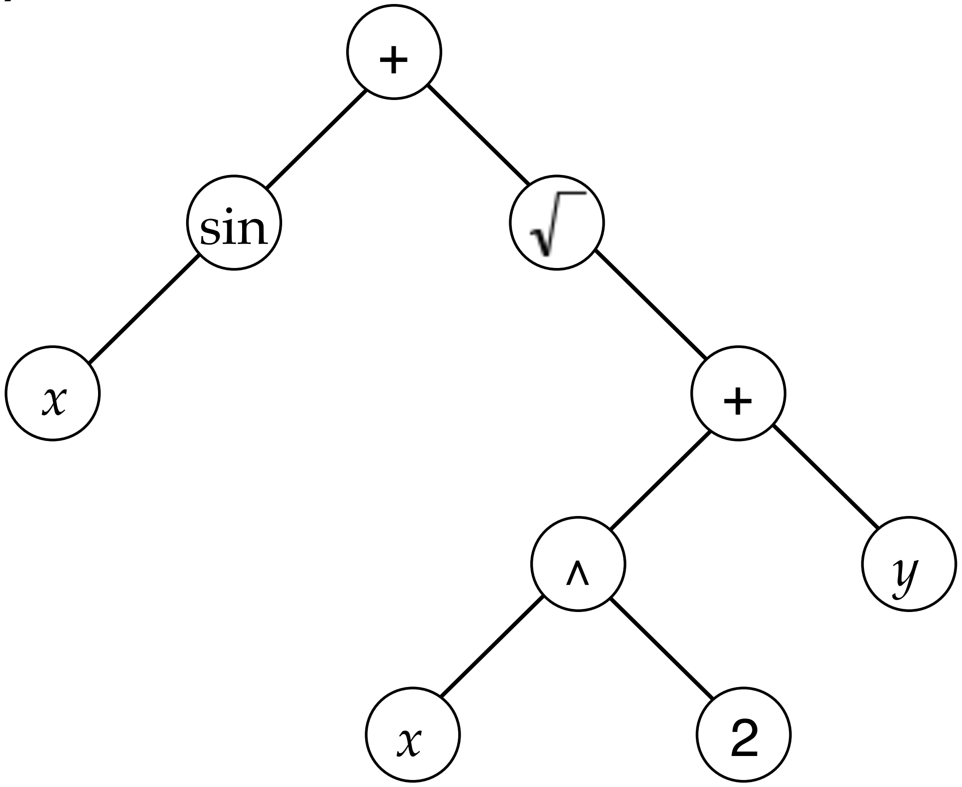


Tournament

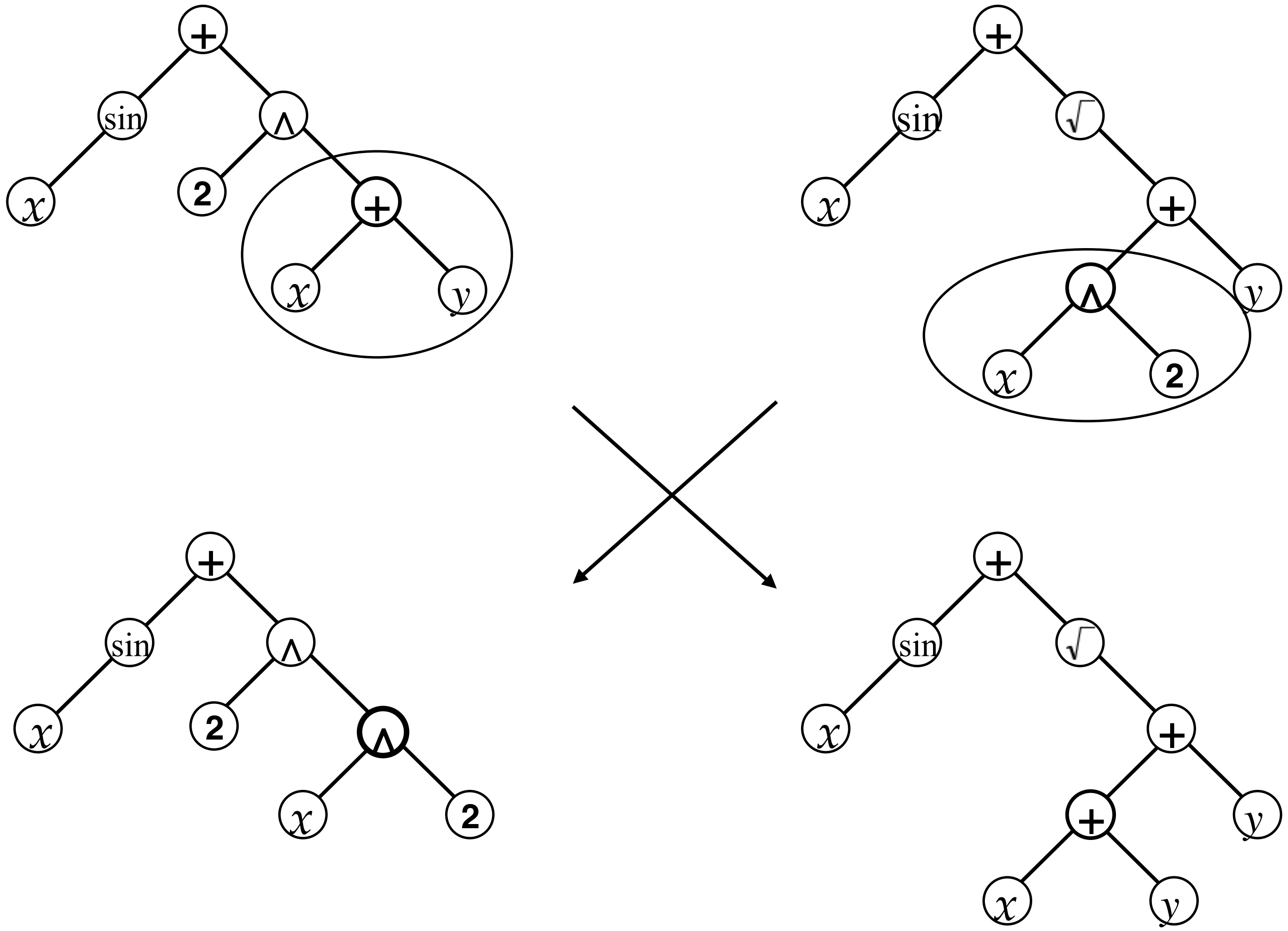
Genetic Programming

Population of programs represented by trees

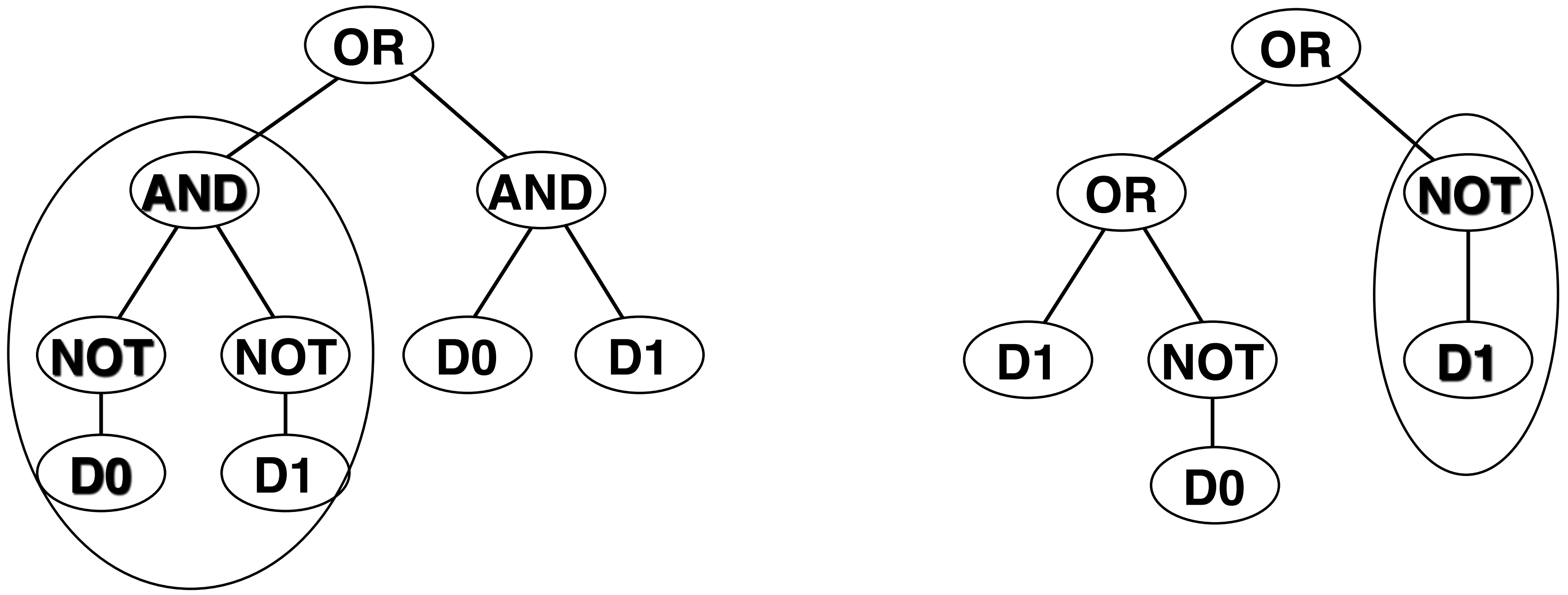
$\sin(x) + \sqrt{(x^2 + y)}$



Crossover

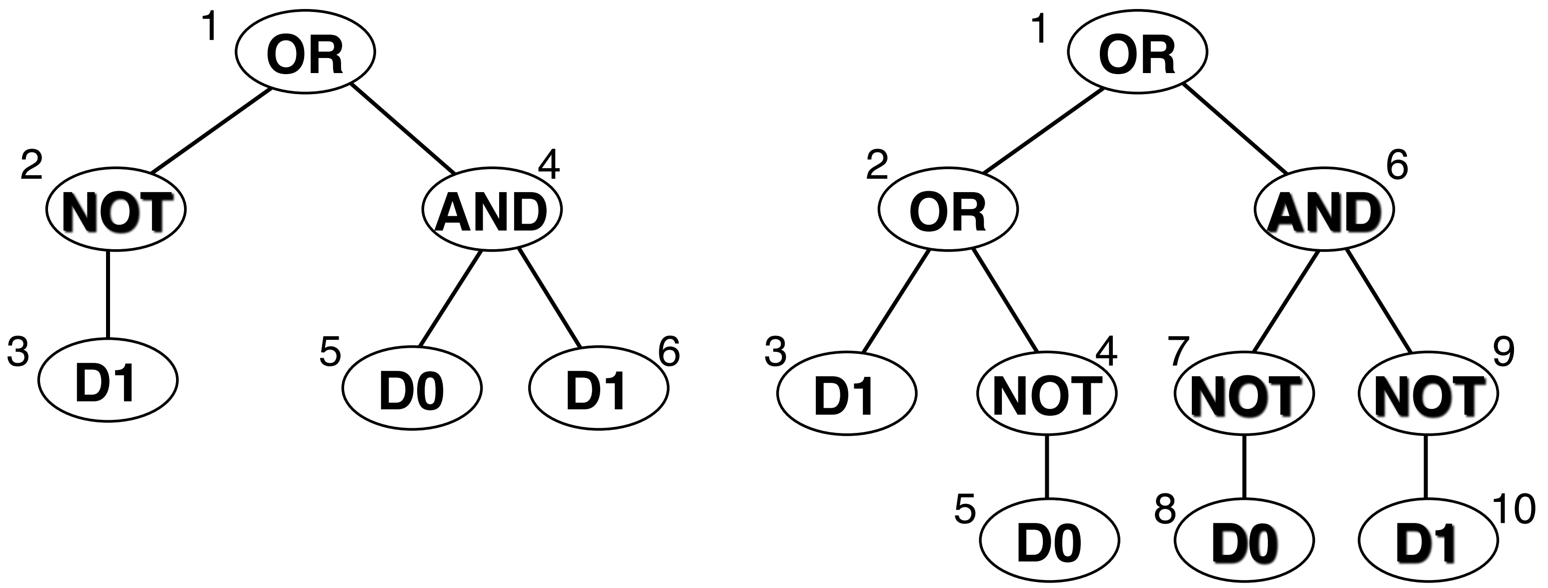


Two Offspring



(OR (AND (NOT D0) (NOT D1))
(AND D0 D1))
(OR (OR D1 (NOT D0))
(NOT D1))

Two Parents in Crossover



(OR (NOT D1)
        (AND D0 D1))
(OR (OR D1 (NOT D0))
        (AND (NOT D0) (NOT D1))

Genetic Programming

More interesting example: design electronic filter circuits

- Individuals are programs that transform beginning circuit to final circuit, by adding/ subtracting components and connections
- Use population of 640,000, run on 64-node parallel processor
- Discovers circuits competitive with best human designs

Summary: Genetic algorithms

- Have been applied to a wide range of problems.
- Results are sometimes very good and sometimes very poor.
- The technique is relatively easy to apply and in many cases it is beneficial to see if it works before thinking about another approach.

Next Lecture

- Another Form of Local Search
  - Repair/Debugging in Constraint Satisfaction Problems
    - GSAT
- A Systematic Approach to Constraint Satisfaction Problems
  - Simple Backtracking Search
  - Informed-Backtracking Using Min-Conflicts Heuristic
  - Arc Consistency for Pre-processing
  - Other approaches to ordering variables and values in search